jamk.fi

WordPressin REST-rajapinnan käyttö Cordova-mobiilisovelluksessa

Mikko Hytönen

Opinnäytetyö
Joulukuu 2016
Mediatekniikan koulutusohjelma
Insinööri (AMK)

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Hytönen, Mikko	Opinnäytetyö, AMK	joulukuu 2016
	Sivumäärä 30	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi
WordPressin REST-rajapinnan käyttö Cordova-mobiilisovelluksessa

Tutkinto-ohjelma
Mediatekniikka

Työn ohjaaja(t)
Pasi Manninen

Toimeksiantaja(t)
Into-Digital Oy

Tiivistelmä

Opinnäytetyö toteutettiin Into-Digital Oy:lle. Toimeksiantajan tavoitteena oli toteuttaa asiakkaalle sivustouudistus ja sen rinnalle mobiilisovellus iOS- ja Android-laitteille. Sovelluksen lähtökohtana oli leikkimielinen tilausosio, jonka rinnalle haluttiin tuoda sivuston sisällöt. Lisäksi sovellukseen päätettiin toteuttaa lisäarvoa tuovia ominaisuuksia, kuten sisällön jakaminen ja ilmoitukset sisältöpäivityksistä sovelluksen käyttäjille.

Sovellus päätettiin toteuttaa Apache Cordova -sovelluskehitysalustalle, joka on tarkoitettu sovellusten kehittämiseen usealle mobiilikäyttöjärjestelmälle yhtäaikaisesti. Cordovalla sovelluskehitys tapahtui moderneilla web-tekniikoilla ja järjestelmien valmiita toimintoja pystyttiin hyödyntämään natiivilaajennusten avulla. Cordovan tukena käytettiin Adoben PhoneGap-kehitystyökaluja, joiden avulla sovellusta ja natiivitoimintoja pystyttiin testaamaan tehokkaasti ja langattomasti lähiverkossa.

Sivustouudistukseen valittiin WordPress-sisällönhallintajärjestelmä, joka toimi myös sovelluksen taustapalveluna sisäänrakennetun rajapinta-arkkitehtuurin ja laajennusten avulla. Sisällönhallintaa laajennettiin vastaamaan asiakkaan tarpeita omilla sisältötyypeillä ja Advanced Custom Fields -laajennuksen tietokentillä.

Lopputuloksena toimeksiantajan asiakas sai tilaamansa uuden sivuston ja mobiilisovelluksen. Toimeksiantajalle toteutus toimii referenssinä ja pohjatoteutuksena vastaaville projekteille. Toteutukseen tehtiin useita jatkossa hyödynnettäviä ratkaisuja, kuten WordPressin rajapinnan käyttöönotto sekä yleispätevä prosessi sen hyödyntämiselle sovelluksissa. Samoin push-viestien käyttöönottoa ja integrointia WordPress-sisällönhallintaan voidaan tarjota jatkossa toimeksiantajan asiakkaille.

Avainsanat (asiasanat)
mobiilisovellukset, Cordova, PhoneGap, WordPress

Muut tiedot

jamk.fi

Description

Author(s)	Type of publication	Date
Hytönen, Mikko	Bachelor's thesis	December 2016
		Language of publication: Finnish
	Number of pages: 30	Permission for web publication: x

Title of publication

Use of WordPress REST API in Cordova mobile application

Degree programme

Media Engineering

Supervisor(s)

Manninen, Pasi

Assigned by

Into-Digital Oy

Abstract

The project was assigned by Into-Digital Oy with the objective to build a new website and a companion mobile app for a client. The app would be available for iOS and Android devices. The main idea for the app was to present a playful ordering instruction section complemented by content from the website. To add value for users, some additional features were planned; e.g. to let users share content and notify them about content updates.

The app was developed using Apache Cordova application development framework, which supports publishing in multiple operating systems. Using Cordova, an app was built with modern web techniques, and native features can be accessed through native plugin architecture and plugins developed by Cordova community. In addition to Cordova, PhoneGap development tools by Adobe were used. The tools enabled testing the app and its native features on iOS and Android devices wirelessly in a local network.

WordPress content management system was used to build the website. The app was connected to WordPress through built-in REST API architecture and plugins. The content management was customized to meet the needs of the client using custom post types and Advanced Custom Fields plugin.

In conclusion, the website and the mobile app were built successfully and the client's requirements were met. For Into-Digital, the project will be used as a reference and as a base implementation for similar projects in the future. Several aspects of the implementation are reusable, e.g. setting up the WordPress REST API and a generic process for accessing it and caching the results for offline use in an app. Another reusable aspect is setting up push notifications and integrating them to WordPress content management.

Keywords/tags (subjects)

mobile applications, Cordova, PhoneGap, WordPress

Miscellaneous

Sisältö

Kuviot

Taulukot

1 Lähtökohdat

1.1 Toimeksiantaja

Työn toimeksiantaja oli Into-Digital Oy, Helsingissä ja Jyväskylässä toimiva digitaalisen median toimisto. Into-Digital tekee pääasiassa toteutuksia markkinointiviestintäalan tarpeisiin: muun muassa sivustoja, teemasivuja, display-mainoksia ja mobiilisovelluksia.

1.2 Tavoitteet

Toimeksiantajan tavoitteena oli toteuttaa asiakkaalle sivustouudistus ja sen rinnalle mobiilisovellus iOS- ja Android-laitteille. Opinnäytetyössä tarkastellaan erityisesti mobiilisovelluksen toteutusta hyödyntäen sivustoa rajapintojen välityksellä.

Toteutettavan sovelluksen lähtökohtana oli leikkimielinen tilausohje, jota käyttäjä pystyy seuraamaan vaihe kerrallaan. Tämän idean rinnalle sovellukseen päätettiin tuoda mukaan sivuston sisällöt sekä toteuttaa sovellukselle lisäarvoa antavia ominaisuuksia, kuten sisältöjen jakaminen ja ilmoitukset sisältöpäivityksistä. Taustalle sovellukseen suunniteltiin seurantamittareita, joiden perusteella sovelluksen käyttäjämääristä, ominaisuuksien käytöstä ja käyttäjäryhmistä saataisiin tilastoja.

Toimeksiantaja on erikoistunut WordPress-sivustoihin ja WordPressin sisällönhallinnan hyödyntämiseen erikoistoteutuksissa. Tästä syystä se oli ilmeinen valinta sivustouudistuksen pohjalle. Sovellus tarjosi myös mahdollisuuden laajentaa yrityksen osaamista WordPressin uuteen, sisäänrakennettuun rajapinta-arkkitehtuuriin.

Toimeksiantajalla oli myös aiempaa kokemusta Apache Cordovasta, jonka päälle sovellus päätettiin toteuttaa. Kokemuksen lisäksi valinnassa oli taustalla mahdollisuus laajentaa myöhemmin Windows-laitteille sekä Cordovan pohjautuminen web-tekniikoihin, mikä on toimeksiantajan ydinosaamista. Toteutus oli toimeksiantajalle ensimmäinen tällä teknologiayhdistelmällä ja toimii sekä referenssinä että pohjatoteutuksena vastaaville projekteille.

2 Apache Cordova

2.1 Yleistä

Apache Cordova on avoimen lähdekoodin sovelluskehitysalusta, joka on ensisijaisesti tarkoitettu sovellusten kehittämiseen usealle mobiilikäyttöjärjestelmälle yhtäaikaisesti. Cordova tarjoaa kullekin tuetulle käyttöjärjestelmälle natiivin rungon sekä rajapinnan natiiveihin toimintoihin ja sensoreihin. Varsinainen kehitys tapahtuu järjestelmäriippumattomasti moderneilla web-tekniikoilla. (Cordova Overview 2016.)

Tällaista mallia kutsutaan hybridisovelluskehitykseksi. Lopullinen sovellus on käyttäjän näkökulmasta täysin natiivi, mutta käyttöliittymää ja logiikkaa ajetaan selainnäkymässä, joka täyttää koko näyttöalan. Sovelluksen saa myös eri valmistajien sovelluskauppoihin. Julkaisuprosessit ovat kuitenkin järjestelmäriippuvaisia, eikä Cordova ota niihin kantaa.

Usealle käyttöjärjestelmälle kehitettäessä tämä kehitysmalli säästää resursseja, yhtenäistää käyttökokemuksen ja päivitykset. Resursseja säästyy, koska ominaisuuksia ei tarvitse toteuttaa useaan kertaan, usealla ohjelmointikielellä. Tästä syystä myös kehitystiimi on mahdollista pitää pienempänä. Web-tekniikoilla toteutettu käyttöliittymä käyttäytyy pitkälti samalla tavalla eri järjestelmissä ja on luonnostaan skaalattavissa eri näyttöko'oille. Koska ominaisuudet toteutetaan kerralla, on päivitykset helpompi julkaista eri käyttöjärjestelmille kerralla. Eri sovelluskauppojen hyväksyntäprosessit voivat tosin viivyttää päivitystä.

Hybridisovellusten kehityksessä on myös haasteita. Päällimmäisenä on saada käyttöliittymä tuntumaan natiivilta ja reagoimaan yhtä nopeasti kuin natiiveissa sovelluksissa. Selaimien nykykehityksen myötä tämä on lähinnä optimointikysymys. Toinen haaste ovat käyttöjärjestelmien eroavuudet. Esimerkiksi käyttöliittymien graafiset ohjeistot voivat olla ristiriitaisia, laitteen fyysiset painikkeet ja sensorit eroavat ja selainten ominaisuudet eivät ole samalla tasolla.

2.2 Tuetut käyttöjärjestelmät

Cordova tukee markkinoiden yleisimpiä mobiilikäyttöjärjestelmiä, joista parhaiten tuettuja ovat iOS ja Android. Windows Phonelle on ollut versiosta 8 lähtien tuki, joka parantui merkittävästi versioissa 8.1 ja 10, kun Microsoft yhtenäisti työpöytä- ja mobiilikehitysalustansa. Lisäksi Cordovalla on mahdollista kehittää työpöytäsovelluksia Windowsille, OS X:lle ja macOS:lle sekä Ubuntulle (ks. taulukko 1).

Taulukko 1. Cordovan tukemat käyttöjärjestelmät (Platform Support 2014)

	Android	**Black-Berry 10**	**iOS**	**Ubuntu**	**Windows Phone 8**	**Windows (8.1, 10, Phone 8.1)**	**OS X, macOS**
cordova CLI	✔ Mac, Windows, Linux	✔ Mac, Windows, Linux	✔ Mac	✔ Ubuntu	✔ Windows	✔	✔ Mac
Embedded WebView	✔	✘	✔	✔	✘	✘	✔
Plugin Interface	✔	✔	✔	✔	✔	✔	✔

2.3 Natiivilaajennukset

Hybridisovelluskehityksen tärkein edellytys on pystyä hyödyntämään samoja toimintoja kuin natiivilla ohjelmointikielellä. Näin hybridisovellus pystyy kilpailemaan natiivisovellusten kanssa ja tarjoamaan käyttäjille lisäarvoa tavalliseen verkkosivuun nähden. Sovelluskaupat voivat jopa valvoa tätä, esimerkiksi Apple voi hylätä sovelluksen joka ei eroa verkkosivusta (App Store Review Guidelines n.d.).

Cordovassa tämä on ratkaistu laajennusrajapinnalla, jonka avulla alustaa pystyy vapaasti laajentamaan. Laajennuksen toiminnot ohjelmoidaan natiiveilla kielillä ja niille määritellään rajapintaan kutsut, joiden kautta toimintoja pystytään käyttämään JavaScriptilla sovelluksen logiikasta. (Cordova Overview 2016.)

Laajennusten toteuttaminen on täysin avointa. Cordova ja sen päälle rakennetut alustat tarjoavat laajennuksia yleisimpiin tarpeisiin, mutta pääosan laajennuksista toteuttaa kehittäjäyhteisö. Laajennusten jakelu tapahtuu pääasiassa julkisten npm- ja git-repositorioiden kautta, joiden kautta asentamista Cordova tukee suoraan.

2.4 Kehitystyökalujen valinta

2.4.1 Johdanto

Cordovaa voi käyttää sellaisenaan alustan mukana tulevilla komentorivityökaluilla. Mutta koska alusta on avoin, sen pohjalta on rakennettu useita kehitystyökaluja ja -malleja. Vaihtoehtoiset työkalut ratkaisevat hybridisovelluskehityksen haasteita osa-alueilla, joihin Cordova ei ota kantaa.

Osa vaihtoehdoista tarjoaa valmiita käyttöliittymäratkaisuja, jotka voivat nopeuttaa natiivin tuntuisen sovelluksen kehittämistä, tai valmiita projektipohjia, joissa on esiasennettuna tiettyyn käyttötapaukseen tarvittavat laajennukset. Esimerkkinä ensin mainitusta, Ionicin ratkaisussa käyttöliittymä toteutetaan AngularJS-alustan päälle ja mukana tulee laaja kirjasto käyttöliittymäkomponentteja (Ionic Component Documentation n.d.). Tällainen on erityisen hyvä ratkaisu, mikäli sovelluksen käyttöliittymästä halutaan myös natiivin näköinen. Jälkimmäisestä esimerkkinä PhoneGap tarjoaa valmiin projektipohjan muun muassa push-viestien käyttöönottoon ja vastaanottamiseen (Push Notifications 2016).

Sovelluksen helppo testaaminen nopeuttaa kehitystä ja vianetsintää. Cordovassa varsinkin natiivien toimintojen testaaminen on hidasta, koska se vaatii sovelluksen kääntämistä ja asentamista alustakohtaisilla työkaluilla. Laitteen tulee myös yleensä olla kytketty tietokoneen USB-liitäntään. Käyttöliittymää on mahdollista testata suoraan työpöytäselaimessa. Esimerkiksi PhoneGap ja Monaca tarjoavat kehittyneempiä testaustyökaluja. Molemmat tukevat testaamista suoraan laitteella verkon yli ja pystyvät päivittämään muutokset toteutukseen automaattisesti. Myös ennalta määrättyjen laajennusten testaaminen on mahdollista suoraan.

Sovelluksen kääntäminen ja julkaiseminen usealle alustalle vaatii normaalisti kehittäjältä paljon työtä ja kokemusta. Kullekin alustalle on omat kehitystyökalunsa, joiden

käyttöönotto ja käyttö tulee opetella. Tähän ratkaisuna on tarjolla palveluita, jotka kääntävät etänä virtuaalikoneilla sovelluksesta julkaisukelpoisen paketin. Esimerkiksi PhoneGapilla ja Monacalla on omat palvelunsa, joihin kehittäjä pystyy lähettämään projektinsa paketoitavaksi.

Lisäksi ratkaisu voi tarjota verkkopalveluita ja rajapintoja tiedon hallintaan ja välitykseen. Esimerkiksi Monaca tarjoaa palveluita muun muassa käyttäjien hallintaan sekä push-viestien ja sähköpostien lähettämiseen selainkäyttöliittymän kautta (Monaca Backend n.d.). Tällöin kehittäjä ei välttämättä tarvitse sovelluksen taustalle lainkaan omaa palvelinlogiikkaa ja tietokantaa.

2.4.2 PhoneGap

Tähän projektiin valittiin pohjaratkaisuksi PhoneGap. PhoneGap on Adoben tuotenimi Cordovalle sekä sitä tukeville työkaluille ja palveluille (Products n.d.). Työkaluihin kuuluvat PhoneGap Desktop app ja PhoneGap Developer app. Maksullisena palveluna Adobe tarjoaa PhoneGap Build -palvelua. PhoneGap ei ota kantaa käyttöliittymän toteutukseen, mikä oli eduksi tässä toteutuksessa. Markkinointiin suunnatun sovelluksen käyttöliittymä haluttiin rakentaa ja optimoida alusta alkaen itse.

PhoneGap Desktop app nopeuttaa projektin perustamista ja kehittämistä verrattuna Cordovan komentorivityökaluihin. Sen käyttöliittymän kautta pystyy perustamaan projektin ja ajamaan paikallista testauspalvelinta. Testauspalvelin käsittelee automaattisesti muutokset lähdekoodiin, jolloin sovellusta ei tarvitse jatkuvasti kääntää testaamista varten. Testauspalvelimeen voi yhdistää työpöytäselaimella tai PhoneGap Developer appilla.

PhoneGap Developer app on mobiilisovellus, joka on ladattavissa sovelluskauppojen kautta iOS-, Android- ja Windows Phone -alustoille. Sen avulla voi yhdistää joko julkisiin PhoneGap-demoihin tai yleisemmin omaan testauspalvelimeen lähiverkossa. Yhdistämisen jälkeen se lataa automaattisesti muutokset testauspalvelimelta. Näin kehittäjä pystyy nopeasti testaamaan sovelluksensa käyttöliittymää natiivissa ympäristössä. Testaussovellus ei tue automaattisesti laajennuksia, mutta tukee ennalta määrättyä joukkoa suosittuja laajennuksia. Esimerkiksi push-viestejä pystyy testaamaan

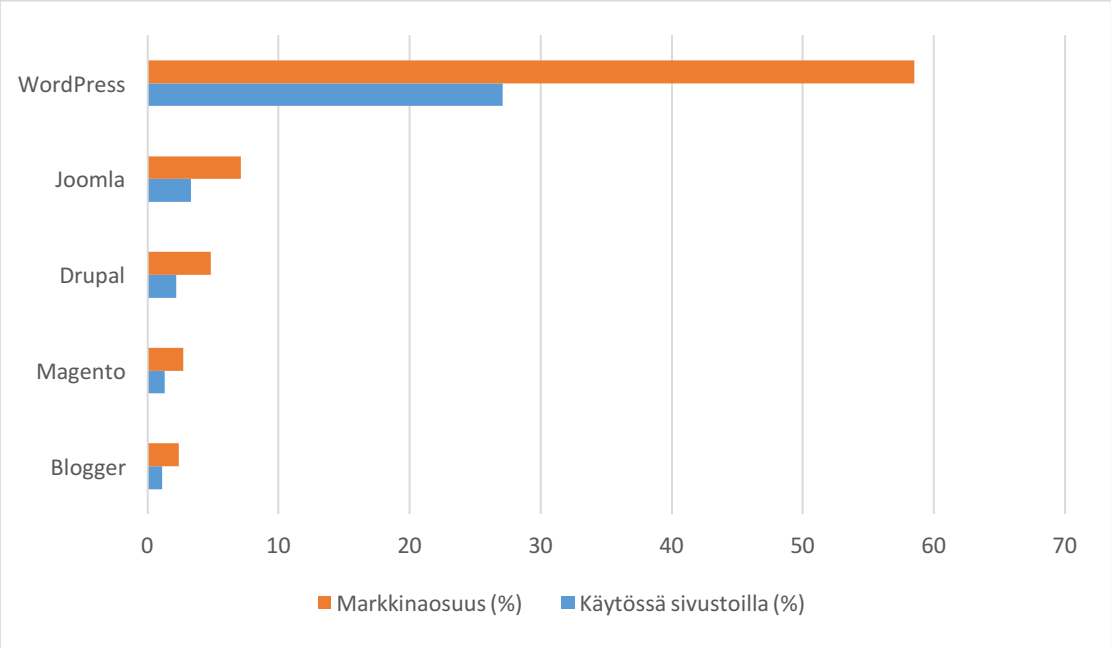
heti, kun ilman testaussovellusta ne vaativat alustakohtaisia, monivaiheisia sertifiointeja ja rajapinta-avaimia sekä palvelimen, joka hoitaa viestien lähettämisen.

PhoneGap Build on maksullinen pilvipalvelu, joka hoitaa sovellusten julkaisemisen ja paketoinnin sovelluskauppoja varten kehittäjän puolesta. Palvelu tukee iOS-, Android- ja Windows Phone -alustoja sekä pitkälti samoja ennalta määrättyjä laajennuksia kuin PhoneGap Developer app. Normaalisti julkaiseminen tapahtuu alustakohtaisilla kehitystyökaluilla, joista osa on lisäksi saatavilla vain tietyille käyttöjärjestelmille. Palvelu siis yhtenäistää ja nopeuttaa julkaisuprosessia varsinkin, jos sovellus julkaistaan useammalle alustalle.

3 WordPressin REST-rajapinta

3.1 WordPress

WordPress on maailman käytetyin verkkosivustojen sisällönhallintajärjestelmä. W3Techsin seurannan mukaan WordPressin markkinaosuus oli 58,5 % sisällönhallintajärjestelmien kesken ja käyttöaste 27,1 % seurattujen sivustojen kesken marraskuussa 2016 (ks. kuvio 1). WordPress on avointa lähdekoodia ja ilmainen, ja sen keskeisiä ominaisuuksia ovat yksinkertainen käyttöliittymä, laajennettavuus yhteisön kehittämien lisäosien kautta ja automaattiset tietoturvapäivitykset.



Kuvio 1. Käytetyimmät sisällönhallintajärjestelmät (Usage of content management systems for websites 2016)

3.2 REST-rajapinta

3.2.1 Johdanto

REST on verkossa yleistynyt malli, joka tarjoaa verkkopalveluille yhtenäisen ja helppolukuisen tavan toteuttaa rajapintoja. Verkossa rajapinta muodostuu hierarkkisista resurssilinkeistä (ks. luku 3.2.2) ja HTTP-pohjaisista operaatioista (Glossary 2015):

- GET-pyyntö hakee yksittäisen tietueen tai joukon tietueita
- PUT-pyyntö korvaa yksittäisen tietueen tai joukon tietueita
- POST-pyyntö lisää tietueen joukkoon
- DELETE-pyyntö poistaa yksittäisen tietueen tai joukon tietueita

Yleisesti REST-rajapintojen kanssa kommunikaatio tapahtuu verkossa joko JSON- tai XML-formaatissa. Malli ei ota tähän kantaa, ja yksittäinen rajapinta voi tukea yhtä tai useampaa formaattia. WordPressin sisäänrakennettu rajapinta tukee oletuksena ainoastaan JSON-formaattia.

Rajapinta on ollut sisäänrakennettuna joulukuussa 2015 julkaistusta versiosta 4.4 lähtien (Mullenweg 2015) ja jo sitä ennen erillisenä lisäosana. Sisäänrakennettu versio 2 tarjoaa kehikon rajapinnoille, mutta julkiset päätepisteet pitää ottaa käyttöön joko

virallisella WordPress REST API (Version 2) -lisäosalla tai toteuttamalla omat päätepisteet projektin tarpeiden mukaan. Lisäosa tarjoaa pääsyn lähes kaikkeen sivuston julkiseen sisältöön, lukuun ottamatta metatietoja (ks. luku 3.2.5). Nämä valmiit päätepisteet pyrkivät olemaan kaiken kattavia, minkä kääntöpuolena omien päätepisteiden toteutus voi olla tarpeen tiedonsiirron ja suorituskyvyn optimoimiseksi.

Rajapinta tukee myös kirjautumista. Kirjautuneen käyttäjän nimissä voidaan hakea myös julkaisematonta sisältöä sekä suorittaa ylläpitotoimenpiteitä. Tälle ei kuitenkaan ollut tarvetta tässä projektissa.

3.2.2 Rajapinnan rakenne

WordPressin tarjoama rajapinta on vahvasti ristiin linkitetty ja itseään dokumentoiva. Rajapinnan juuripäätepisteestä osoitteessa /wp-json/ on haettavissa kaikki rajapinnan nimiavaruudet ja niiden määrittelemät päätepisteet, mukaan lukien kehittäjän itse lisäämät.

WordPress REST API (Version 2) -lisäosa käyttää nimiavaruutta /wp/v2. Se noudattaa pitkälti WordPress-sivuston rakennetta:

- /types: sisältötyypit, oletuksena post, page ja media
- /statuses: sisältöjen tilat, oletuksena publish eli julkinen
- /taxonomies: sisältöjen taksonomiat, oletuksena category ja post_tag
- /comments: käyttäjien kommentit sisällöille
- /[slug]: tietyn sisältötyypin sisällöt tai taksonomian termit ovat haettavissa slug-nimillä

Slug on WordPressin termi verkko-osoitteissa käytettäville tunnisteille. Ne voivat sisältää vain pieniä kirjaimia ja tarvittaessa väliviivoja sanojen välissä. Esimerkiksi sivuston kaikki sivut on mahdollista hakea päätepisteeltä /pages, eli osoitteesta /wp-json/wp/v2/pages.

3.2.3 Sivuston rakenne

Tässä projektissa WordPressin sisältörakennetta laajennettiin omilla sisältötyypeillä ja taksonomioilla. Määrittelemällä sisältörakenne asiakkaan tarpeiden mukaan sel-

keyttää sekä sisällönhallintaa että rajapinnan toteutusta ja käyttöä. Omat sisältötyypit lisättiin sivuston eri sisältöosioille: Galleria (gallery), Tuotteet (products) ja Historia (history). Galleriaa varten luotiin media-taksonomia materiaalien luokitteluun.

Myös sisäänrakennettuja sisältötyyppejä käytettiin niiden soveltuessa tarkoitukseen. Yksittäisiin tekstisivuihin käytettiin valmista page-sisältötyyppiä. Käyttämätön post-sisältötyyppi piilotettiin kokonaan sisällönhallinnasta.

Uudet sisältötyypit ja taksonomiat lisätään luontivaiheessa rajapinnalla /wp/v2-nimiavaruuteen antamalla rekisteröintifunktiolle argumentti show_in_rest arvolla true.

```
register_post_type('gallery',
    array(
        …
        'show_in_rest' => true,
        'rest_base' => 'gallery'
    )
);
register_taxonomy('media',
    array('gallery'),
    array(
        …
        'show_in_rest' => true,
        'rest_base' => 'medium'
    )
);
```

3.2.4 Advanced Custom Fields -laajennus

Projektissa käytettiin kattavasti Advanced Custom Fields -laajennusta. ACF:n avulla sivuston ylläpitäjä pystyy laajentamaan ja jäsentelemään sisällönhallintaa uusilla sisältökentillä ja -rakenteilla. Hyvin suunniteltuna se helpottaa ylläpitoa ja eriyttää ylläpidettävän sisällön sen julkisesta, visuaalisesta esittämistavasta. ACF tallentaa kuitenkin tiedot yhtenäisesti WordPressin metatietoihin. Laajennuksen perusversio on ilmainen, ja siitä on olemassa pro-versio, joka tarjoaa useita hyödyllisiä lisäominaisuuksia. (Condon 2016.)

Koska metatiedot eivät ole saatavilla julkisten REST-päätepisteiden kautta (ks. luku 3.2.5), ACF-tiedot tulee joko hakea itse tai lisätä automaattisesti julkisiin päätepisteisiin ACF to REST API -laajennuksen avulla.

ACF to REST API lisää automaattisesti ACF-sisällöt osaksi /wp/v2-nimiavaruutta. Esimerkiksi tuoteartikkelin tiedoista löytyy acf-niminen tietue, jonka alta löytyy tuotteelle liitetyt lisätiedot, kuten valmistusaineet ja ravintosisältö. Lisäosa tarjoaa myös oman /acf/v2-nimiavaruuden yksittäisten tietojen hakemiseen ja muokkaamiseen.

3.2.5 Rajoitukset

REST-rajapinnan olennaisin rajoitus on metatietojen käsittelyssä. Julkisten päätepisteiden kautta metatietoihin ei pääse lainkaan, koska ne voivat julkisen sisällön lisäksi sisältää ylläpitäjien yksityisiä ja sivuston suojattuja tietoja. Lisäksi metatiedot voivat sisältää serialisoitua dataa, jonka käsittely on estetty myös teknisistä ja tietoturvasyistä. (Common Problems 2016.)

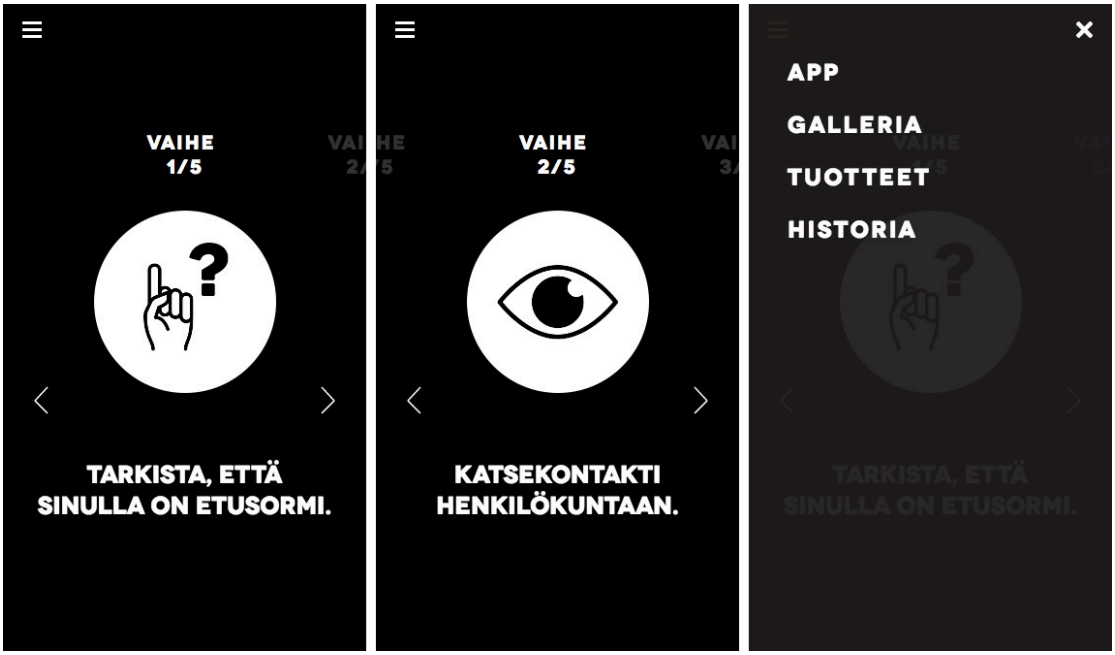
Näitä rajoituksia on mahdollista kiertää harkiten valmiilla laajennuksilla ja itse tehdyillä päätepisteillä. Jälkimmäisessä vaihtoehdossa haettavat tiedot on myös mahdollista rajoittaa toteutuskohtaisesti.

4 Mobiilisovellus

4.1 Käyttöliittymä

4.1.1 Rakenne

Sovellukseen saatiin asiakkaalta rautalangat ja graafiset elementit. Mobiilisovelluksissa käytetään yleisesti kiinteää latauskuvaa käynnistyksen aikana, tässä tapauksessa käytettiin tasaisen mustaa kuvaa. Käynnistymisen jälkeen mustalle taustalle animoitiin logo, käyttöliittymäelementit ja aloitusnäkymä. Sovellus alkaa tilausosiosta, jossa käyttäjä voi seurata vaihe vaiheelta ohjeita tilaamiseen. Vaiheiden välillä liikutaan joko sivulle pyyhkäisemällä, nuolipainikkeista tai otsikoista napauttamalla. Valikkopainikkeen kautta pääsee muihin sovelluksen osioihin: galleria, tuotteet ja historia (ks. kuvio 2).



Kuvio 2. Sovelluksen käyttöliittymä: tilausosio ja valikko

Galleria-osiossa listataan allekkain asiakkaan markkinointimateriaalia. Listausta voi rajata median tyypin mukaan, esimerkiksi YouTube-videoihin, printtimainoksiin tai ulkomainoksiin. Galleria on sovelluksen aktiivisin osio, sen sisältö päivittyy säännöllisesti ja käyttäjillä on mahdollisuus jakaa gallerian materiaalia. Materiaalin ohessa näytetään käyttöjärjestelmän graafisen ohjeiston mukainen jakopainike, joka avaa käyttöjärjestelmän jakotoiminnon (ks. luku 4.5).

Tuotteet esitetään pyyhkäisemällä navigoitavina tuotekortteina, samalla rakenteella kuin tilausosion vaiheet. Historia-osiosta löytyy aikajana asiakkaan brändille merkittävistä vuosista. Vuodet esitettiin janalla visuaalisesti siten, että ne vuorottelevat janan vasemmalla ja oikealla puolella, lähtien alkuvuosista ja päättyen nykyhetkeen.

4.1.2 Valitut HTML5-tekniikat ja JavaScript-kirjastot

Sovelluksen käyttöliittymä toteutettiin JavaScript-pohjaisesti, jotta kaikki sen osa-alueet olivat optimoitavissa ja jotta se saatiin vaikuttamaan mahdollisimman natiivilta. Käyttöliittymäkirjastoksi valittiin Ractive.js, joka sitoo sisällön eli datan HTML-pohjiin, joista käyttöliittymä koostuu (Harris 2016). Tämä tarkoittaa, että muutokset sisältöön päivittyvät automaattisesti käyttöliittymään ja sovelluslogiikan tarvitsee huolehtia vain datasta tietämättä mitään sen esitystavasta.

Lisäksi käyttöliittymän osa-alueiden toteutuksessa käytettiin Ractivea tukevia JavaScript-kirjastoja. Kosketusnäyttöjen kosketusten käsittelyyn käytettiin Hammer.js-kirjastoa, joka muuttaa kosketusnäytöiltä saatavan datan helposti käsiteltäviksi tapahtumiksi, kuten napautuksiksi ja pyyhkäisyiksi. Hammer.js on integroitu Ractiveen ractive-touch-laajennuksella (Cruz 2015). Yleinen käyttötapaus sovelluksessa oli sisällön vaihtaminen pyyhkäisemällä sivuille. Tähän käytettiin Slick-karusellikomponenttia, joka hoitaa sivuttaissiirtymät sulavasti, skaalautuu näyttökoon mukaan ja on ohjattavissa JavaScriptilla (Wheeler 2016). Näiden lisäksi käytettiin jQuery-yleiskirjastoa viestintään komponenttien välillä ja paikkaamaan osa-alueita, joihin edellä mainitut eivät soveltuneet ja jotka olisivat olleet työläämpiä toteuttaa pelkällä JavaScriptilla.

Datan säilyttämiseen ja käyttäjien valintojen muistamiseen käytettiin selaimiin sisäänrakennettua LocalStorage-rajapintaa.

4.1.3 Ractive.js-käyttöliittymä

Ractive.js:llä käyttöliittymän voi koostaa komponentteina, jotka liitetään tavalliseen HTML-dokumenttiin. Yksittäinen komponentti koostuu HTML-pohjasta, datasta ja tapahtumien käsittelystä. Tavallisen HTML:n lisäksi Ractiven pohjissa käytetään laajennettua mustache-syntaksia datan sitomiseen sekä omia attribuutteja tapahtumien käsittelyyn ja siirtymäanimaatioiden kiinnittämiseen (Harris 2016). Yksinkertaisin tapa toteuttaa pohja on sisällyttää se HTML-dokumenttiin script-elementtinä, johon voidaan viitata id-attribuutilla.

```
<script id="order-template" type="text/ractive">

  <main id="main" class="site-main swipe-navigation" role="main">

    {{#each instructions}}

    <article id="order-{{index}}" class="step">
      <div class="entry-content">
        <h3>{{instruction}}</h3>
        {{#graphic}}
        <div class="graphic"><img class="wp-post-image"
src="{{graphic}}" alt="" on-load="downsize" /></div>
        {{/graphic}}
      </div>
    </article>

    {{/each}}

  </main>

  <button on-tap="previousstep" class="previous-step step-ar-
row"></button>
  <button on-tap="nextstep" class="next-step step-arrow"></but-
ton>

</script>
```

Kaikki mustache-tyylinen syntaksi on kahden aaltosulun sisässä ja ensin voi tulla operaatio, jota seuraa aina muuttuja. Esimerkkipohjassa instructions on taulukko, jonka jokaiselle tietueelle luodaan #each-operaatiolla article-elementti. Mikäli tietueella on kuva graphic-muuttujassa, luodaan sille elementti #-operaatiolla, joka vastaa if-ehtolauseketta. Mikäli aaltosuluissa on ainoastaan muuttuja ilman operaatiota, muuttujan sisältö tulostetaan tekstinä. Kolmella aaltosululla on mahdollista tulostaa HTML-sisältöä.

Lisäksi pohjassa on käytetty on-load- ja on-tap-attribuutteja tapahtumien kiinnittämiseen. Attribuutin arvona annetaan tapahtuman nimi, toisin kuin perinteisissä JavaScript-attribuuteissa, joissa arvo on suoritettavaa ohjelmakoodia. Pohja otetaan käyttöön JavaScript-logiikassa luomalla uusi Ractive-instanssi. Allaolevassa esimerkkikoodissa luodaan instanssi, jolle annetaan argumenttina aiemman HTML-pohjan id ja id HTML-elementille, johon instanssi kiinnitetään. Viimeisenä argumenttina annetaan data-muuttuja, joka välitetään HTML-pohjalle. Muutokset data-muuttujaan peilataan

automaattisesti pohjaan. Instanssin luomisen jälkeen siihen kiinnitetään tapahtumien käsittelijät pohjassa annetuilla nimillä.

```
app.view.order = new Ractive({
    template: '#order-template',
    el: '#order-view',
    data: {
        instructions: [],
    }
});
app.view.order.on('downsize', app.downsize);
app.view.order.on('nextstep', app.nextstep);
app.view.order.on('previousstep', app.previousstep);
```

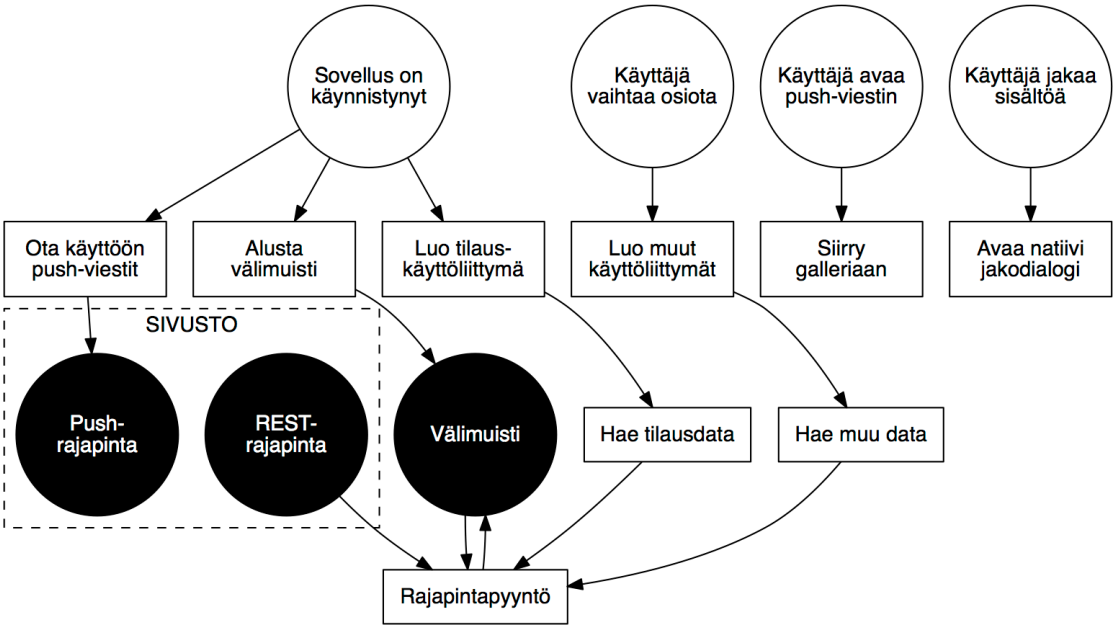
Ractive soveltui projektiin hyvin, koska se oli helppo sovittaa erilliseen logiikkaan ja HTML-pohjiin pystyttiin hyödyntämään WordPress-sivuston lähdekoodeja. Käyttämällä Ractiven tap-tapahtumia JavaScriptin click-tapahtumien sijasta pystyttiin myös kiertämään noin 300 millisekunnin viive tapahtuman suorittamisessa. Oletuksena kosketusnäyttölaitteilla on selainpohjaisissa toteutuksissa viivettä, koska selaimen pitää odottaa suorittaako käyttäjä tuplanapautuksen. Sovelluksessa viiveen kiertäminen on olennaista, koska viive saa sovelluksen käyttöliittymän vaikuttamaan laiskalta.

4.2 Sovellusarkkitehtuuri

Sovellus lähtee liikkeelle Cordovan perusrakenteella. HTML-dokumentin lopussa ladataan JavaScript-kirjastot: ensin Cordovan oma, sitten kehittäjän valitsemat ja lopuksi kehittäjän oma logiikka. Näiden latauduttua kiinnitetään käsittelijät keskeisille tapahtumille, kuten verkkoyhteyden muutoksille, sovelluksen sulkemiselle ja uudelleen avaamiselle. Tässä vaiheessa Cordovan laajennukset eivät ole vielä käytössä ja logiikan tulee odottaa deviceready-tapahtumaa Cordovalta.

Kun Cordova on ladannut laajennukset, voidaan varsinainen sovelluslogiikka käynnistää. Ensimmäisenä pyritään alustamaan sovelluksen välimuisti offline-käyttöä varten (ks. luku 4.3), otetaan käyttöön push-viestit (ks. luku 4.4) ja luodaan tilausosion käyttöliittymäinstanssi. Seuraavaksi käyttöliittymälle haetaan sisällöt keskitetyllä rajapintapyyntömetodilla. Metodi hakee ensin tilausosion datan välimuistista ja palauttaa sen välittömästi callback-metodille, jotta käyttöliittymä on heti käytettävissä. Sitten

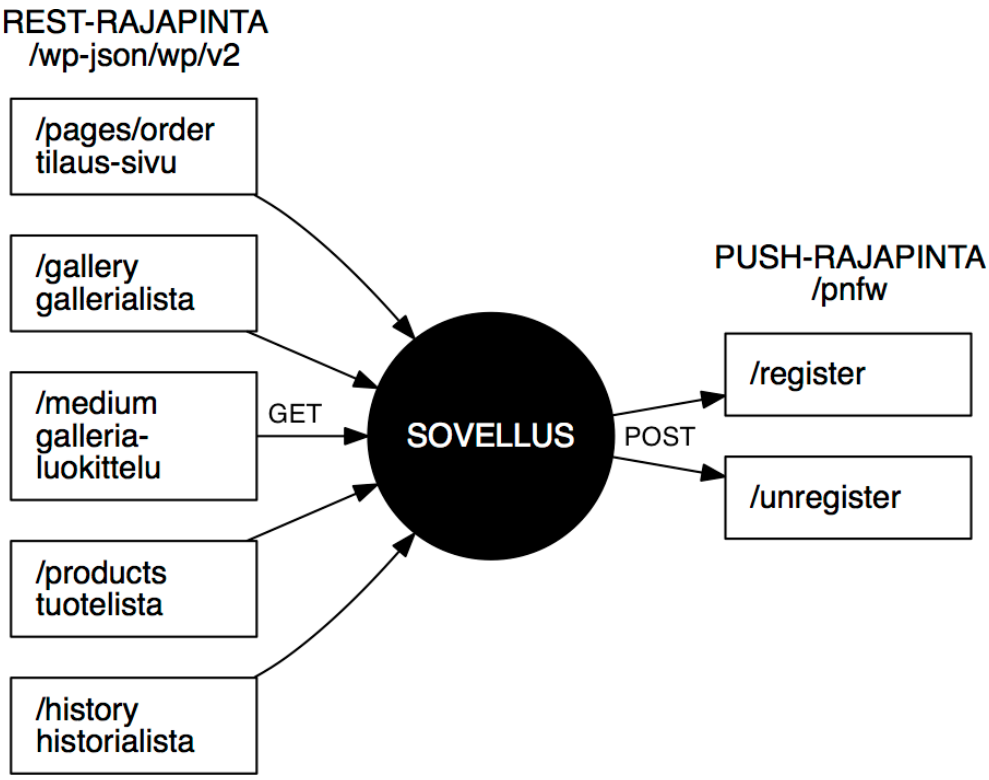
metodi välittää pyynnön REST-rajapinnalle ja saatuaan vastauksen päivittää välimuistin ja palauttaa päivitetyn datan (ks. kuvio 3). Data viedään käyttöliittymäinstanssille, joka peilaa muutokset automaattisesti käyttöliittymään.



Kuvio 3. Sovelluksen arkkitehtuuri

Käyttäjän vaihtaessa osiota suoritetaan sama prosessi kaikille lopuille osiolle yhtä aikaa. Ajatuksena oli, että yksi hieman pidempi lataus tarjoaa paremman käyttökokemuksen kuin lataaminen jokaisen osion kohdalla. Osioille luodaan siis käyttöliittymäinstanssit ja haetaan sisällöt rinnakkaisilla rajapintapyynnöillä (ks. kuvio 4). Logiikka pitää kirjaa suoritetuista pyynnöistä ja tekee vielä taustalla toimenpiteitä, kun kaikkiin pyyntöihin on saatu vastaus.

Mikäli käyttäjä on hyväksynyt push-viestit, tulee galleriapäivityksistä ilmoitus. Jos sovellus on käynnissä, niin näytetään ilmoitus sovelluksessa, muussa tapauksessa se näkyy käyttöjärjestelmän tilariville ja mahdollisesti lukitusnäkymässä. Jos käyttäjä avaa ilmoituksen, avautuu sovelluksen gallerianäkymä, jossa uusi sisältö on nähtävissä. Galleriassa käyttäjä voi myös jakaa sisältöä natiivilla jakodialogilla (ks. luku 4.5).



Kuvio 4. Sovelluksen rajapintapyynnöt

4.3 Offline-käyttö

4.3.1 Johdanto

Sovelluksen keskeiset sisällöt tuli olla käytettävissä myös ilman verkkoyhteyttä. Tähän tarkoitukseen toteutettiin LocalStorage-pohjainen paikallinen tietokanta ja synkronointiprosessi, joka hoitaa säännöllisesti muuttuvan sisällön päivittymisen.

Lisäksi sovellukseen sisällytettiin siemendataa, jolla tietokanta alustetaan. Näin sovelluksen tärkeimmät osiot ovat käytettävissä, vaikkei laitteella olisi ensimmäisellä käynnistyksellä verkkoyhteyttä. Ja käänteisesti tärkeimmätkin osiot ovat päivitettävissä, koska ne käyttävät alustettua tietokantaa sen sijaan, että data olisi kiinteästi upotettuna lähdekoodiin. Alustaminen tapahtuu ensimmäisellä käynnistyskerralla ja tarvittaessa sovelluspäivityksen jälkeen.

4.3.2 LocalStorage-tietokanta

LocalStorage on yksinkertainen lista avain-arvo-pareja ja soveltuu REST-rajapinnan pyyntöjen pitkäaikaiseen tallentamiseen. Avaimena toimii rajapintapyynnön osoite, ja arvoksi tallennetaan JSON-muotoinen merkkijono. Näin LocalStoragea voi käyttää

saumattomasti rajapintapyyntöjen välimuistina, eikä varsinaisen logiikan tarvitse tietää, saatiinko pyydetty data palvelimelta vai suoraan välimuistista.

LocalStorage soveltuu kuitenkin vain rajalliselle määrälle tietoa. Yhden sivuston tai sovelluksen käyttämä tila on rajoitettu, pienimmillään 5 megatavuun. Lisäksi LocalStoragen suorituskyky huononee merkittävästi suurilla määrillä tietoa. Tällöin kannattaa harkita laajennusten käyttöä, joilla voi tallentaa vapaasti laitteen tiedostojärjestelmään tai esimerkiksi LocalStoragea tehokkaampaan SQLite-tietokantaan. (Storage 2016.)

Tämän projektin tarpeisiin LocalStoragen tila on täysin riittävä, eikä rajapinnan välimuistina käyttö vaadi suorituskykyyn vaikuttavia toimenpiteitä. Eri rajapintapyyntöjä on rajallisesti, eikä niiden määrä kasva ajan myötä. JSON-merkkijonojen käsittely on ainoa kuormittava toimenpide, mutta se suoritetaan rajapintapyynnöille välimuistiratkaisusta riippumatta.

4.3.3 Online-synkronointi

Sovellus pyrkii päivittämään tietokannan tiedot jokaisella käynnistyskerralla, jos laitteella on verkkoyhteys. Kaikkea tietoa ei kuitenkaan päivitetä käynnistyksen yhteydessä, vaan tarpeen mukaan. Ainoastaan aloitusnäkymän tiedot pyritään hakemaan heti käynnistyksessä. Muiden osioiden tiedot haetaan, kun käyttäjä siirtyy niihin navigaation kautta. Näin synkronointi vaikuttaa minimaalisesti suorituskykyyn aloitusnäkymässä. Muissa osioissa taas voi olla havaittava latausviive, joka kuitenkin minimoidaan käyttämällä paikallisen tietokannan tietoja jo synkronoinnin aikana. Tämä optimointi noudattaa myös asiakkaan prioriteetteja: heille aloitusnäkymä on tärkein osio, ja muu sisältö on vain sitä tukemassa.

Varsinainen synkronointiprosessi on erittäin kevyt. Rajapinnalta ladataan vain muutama kilotavu tietoa aloitusnäkymään, ja kehitysvaiheessa kaikkien osioiden tiedonsiirto oli alle 20 kilotavua. Siirtomäärään vaikuttaa merkittävästi, että rajapintapalvelimella on käytössä pyyntöjen gzip-pakkaus. Pakkaus pienentää merkittävästi tekstimuotoisen tiedon siirtoa, esimerkkitapauksessa tiedonsiirto oli vain 16 % pakkaamattomasta datasta. Synkronoinnin lisäksi liikennettä voi kuitenkin tulla kuvien ja videoiden lataamisesta sivustolta ja ulkopuolisilta tarjoajilta.

4.4 Push-viestit

4.4.1 Johdanto

Sovelluksen keskeinen ominaisuus on vastaanottaa push-viestejä sisältöpäivityksistä. Push-viesti tarkoittaa sovelluskohtaista ilmoitusta, joka tulee käyttäjän laitteelle riippumatta siitä, onko sovellus käynnissä. Käyttäjä myös kontrolloi, mitä viestejä haluaa vastaanottaa. Näistä syistä viestiarkkitehtuuri on useimmiten toteutettu käyttöjärjestelmän tasolla ja viestit kulkevat keskitetyn välityspalvelun kautta.

4.4.2 Välityspalvelut

Välityspalvelut ovat pitkälti käyttöjärjestelmäkohtaisia. iOS-järjestelmään on integroitu Apple Push Notification Service (APNS) ja Androidille on tällä hetkellä Google Cloud Messaging (GCM) ja sen vähitellen korvaava Firebase Cloud Messaging (FCM). Molemmat Googlen palvelut kuitenkin tukevat myös iOS-laitteille lähettämistä välittämällä viestejä edelleen APNS:lle.

4.4.3 Käyttöönotto

Koska arkkitehtuuri ei ole aivan yksinkertainen ja rajapintoja on useita, on push-viesteihin tarjolla lukuisia laajennuksia ja kolmannen osapuolen palveluita. PhoneGapilla on viesteille oma laajennus ja tuki niiden testaamiselle kehitystyökaluissa. Laajennus hoitaa alustariippumattomasti laitteiden rekisteröinnin välityspalveluihin ja viestien vastaanottamisen. WordPressille taas on Push Notifications for WordPress -laajennus, joka hoitaa käyttäjätunnisteiden tallentamisen ja viestien lähettämisen käyttäjille. Kehittäjän vastuulle jää vielä välityspalveluiden käyttöönotto.

Välityspalvelut vaativat käyttöönotossa sovelluksen rekisteröimisen, jolloin kehittäjä saa pääsyoikeuden palvelun rajapintaan. APNS:ää varten tämä vaatii sertifikaattitiedoston luomista ja push-oikeutuksen liittämistä sovelluksen profiiliin Applen Developer-sivustolla (Adding Capabilities 2016). Googlelle taas ilmoitetaan sovelluksen tunniste ja palvelimen osoite, jota vastaan saadaan rajapinta-avain ja sovelluskohtainen tunniste (Set up a GCM Client App on Android 2016). Sertifikaatti ja rajapinta-avain tallennetaan WordPress-laajennuksen asetuksiin ja sovelluskohtainen tunniste PhoneGap-laajennuksen asetuksiin.

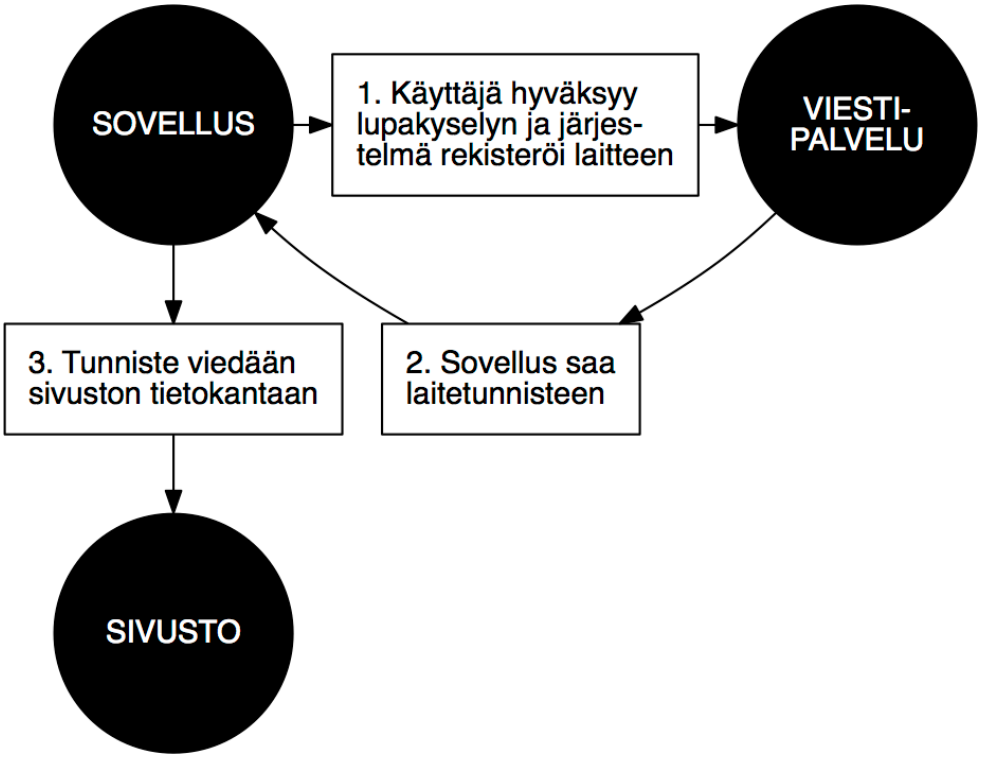
4.4.4 Käyttäjän rekisteröinti

Ensimmäiseksi käyttäjältä pyydetään sovelluksessa lupa lähettää push-viestejä, jolloin laite rekisteröidään sovelluksen tunnisteelle välityspalvelussa. Sovellus saa välityspalvelulta käyttäjätunnisteen, joka tulee tallentaa myöhempää käyttöä varten esimerkiksi palvelimen tietokantaan (ks. kuvio 5).

```
// Laajennuksen käyttöönotto ja lupakyselyn
// esittäminen ensimmäisellä käyttökerralla
var push = PushNotification.init({
    "android": {
        "senderID": "453044274800",
        "icon": "notification"
    },
    "ios": {
        "sound": true,
        "vibration": true,
        "badge": true
    },
    "windows": {}
});

// Laitteen rekisteröinti onnistui
push.on('registration', function(data) {
    // data.registrationId sisältää laitteen tunnisteen,
    // joka lähetetään WordPress-laajennukselle talteen
});

// Laitteen rekisteröinti epäonnistui, yleensä johtuen
// virheellisistä asetuksista kehitysvaiheessa
push.on('error', function(e) {
});
```

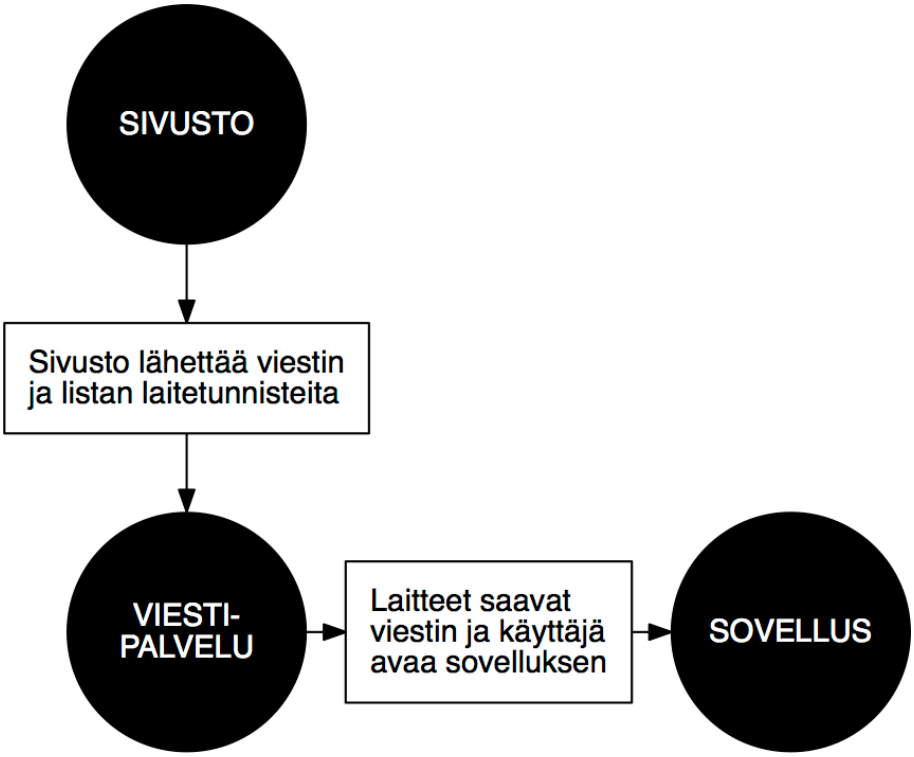


Kuvio 5. Käyttäjän rekisteröiminen push-viestien välityspalveluun

4.4.5 Käyttö

Kun käyttäjille halutaan välittää viesti, se lähetetään sivustolta käyttäjätunnisteiden kanssa välityspalvelulle, josta käyttäjän laite saa viestin. Jos käyttäjä valitsee viestin, avaa käyttöjärjestelmä välityspalvelimelle rekisteröidyn sovelluksen ja välittää sille viestiin kiinnitetyn datan (ks. kuvio 6).

```
push.on('notification', function(data) {
    // Käyttäjä sai viestin ja aktivoi sen,
    // data sisältää tietoa WordPress-laajennukselta
});
```



Kuvio 6. Push-viestien lähettäminen

4.5 Natiivijaot

Jakotoimintoa varten otettiin käyttöön kolmannen osapuolen socialsharing-laajennus. Sen toimintaperiaate on hyvin yksinkertainen: laajennukselle lähetetään otsikko, viesti ja jaettava tiedostot tai verkko-osoite, jolloin se avaa käyttöjärjestelmän jako-dialogin ja käyttäjä voi jakaa sisällön millä tahansa jakamista tukevalla sovelluksellaan. Käytännössä toteutus vaatii kuitenkin tasapainoilua käyttöjärjestelmien ja sovellusten käyttäytymisen mukaan. (Verbruggen 2016.)

Sovelluksessa jakotoiminto kiinnitettiin galleriaosion kaikkien materiaalien yhteyteen (ks. kuvio 7). Koska otsikkotietoa käyttivät vain harvat testatuista jakosovelluksista, eikä varsinaista pidempää viestiä ollut, käytettiin otsikkona ja viestinä samaa tekstiä. Jos jaettava sisältö oli YouTube-video, niin laajennukselle vietiin videon verkko-osoite url-muuttujassa. Useimmat testatuista sovelluksista osasivat upottaa videon suoraan jakoon tai näyttää esikatselun, josta aukeaa YouTube-sovellus. Muissa tapauksissa videolinkki aukeaa vähintään laitteen selaimeen. Kuvien tapauksessa laajennukselle vietiin kuvan osoite files-taulukkoon ja url-muuttuja jätettiin tyhjäksi. Testatut sovellukset jakoivat kuvan luotettavasti vain, jos jakoon ei liitetty erillistä jako-osoitetta.

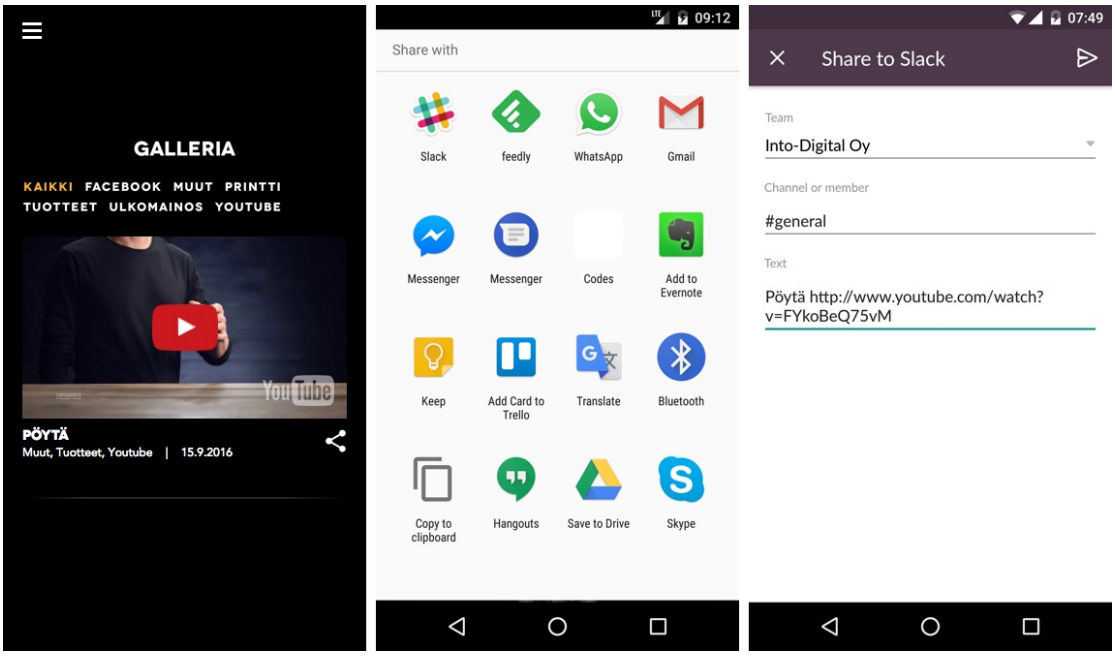
Tällöin kuva tuli lähes aina näkyviin suoraan jakoon, olipa kyseessä multimediaviesti tai sosiaalisen median kanava.

```
    function share (event, subject, image, url) {
        if ('socialsharing' in window.plugins) {

            var options = {
                subject: subject,
                message: subject,
                url: url,
                files: null
            };

            if (image) {
                options.files = [image];
            }

            window.plugins.socialsharing.shareWithOptions(options);
        }
},
```



Kuvio 7. Käyttöjärjestelmän jakodialogi Androidilla

4.6 Seuranta

Sovelluksen asennusten seuranta onnistuu sovelluskauppojen hallintapaneeleista, joista näkee asennusten kokonaismäärän ja aikajanan asennuksista päivän tarkkuudella. Tarkempaa käytön seurantaa varten sovellukseen asennettiin google-analytics-

laajennus. Lisäksi harkittiin Facebook-laajennuksen ja sen seurannan käyttämistä, mutta sille ei lopulta koettu tarvetta.

Seurantaa varten sovellus tulee rekisteröidä mobiilisovelluksena Google Analytic-sissa. Käyttöönottoa varten saadaan vastaava tunniste (Tracking Id) kuin verkkosivus-ton seurantaan. Sovelluksen osioiden välillä navigointi seurataan sivulatauksina ja kaikki muut toiminnot tapahtumina. Tilausohjetta seurataan tarkimmin, jotta pysty-tään mittaamaan käyttäjien kiinnostusta. Ensin mitataan aloittaako käyttäjä ohjeen käytön siirtymällä ensimmäisestä vaiheesta eteenpäin. Jokainen vaihe kirjataan jär-jestysnumeron mukaan, jotta nähdään kuinka pitkälle käyttäjät keskimäärin selaavat ohjetta ja kuinka monta prosenttia jatkaa seuraaviin vaiheisiin. Jos käyttäjä seuraa ohjetta loppuun saakka, kirjataan se lopuksi omana tapahtumaan. Tilausosion lisäksi seurataan käyttäjän liikkumista tuotekorttien välillä sekä paljonko käyttäjät jakavat gallerian materiaalia ja mitä materiaalia jaetaan eniten.

Tässä projektissa Facebookin seurannasta luovuttiin, mutta sille voi olla käyttöä jat-kossa. Analyticsin tapaan sovellus tulee rekisteröidä Facebook-sovellukseksi. Tämän jälkeen laajennus otetaan käyttöön Facebookin antamilla tunnisteilla ja seurannan perusominaisuudet toimivat automaattisesti. Facebook mittaa sovelluksen asennus-ja käynnistyskerrat sekä käyttäjien demografiatietoja, kuten sukupuoli- ja ikäja-kaumaa. Facebook pystyy keräämään demografiatietoja, koska se saa tietoa käyttä-jistään, jos näillä on samalla laitteella asennettuna myös Facebook-sovellus. Jatkossa kerättyjä tietoja pystyy käyttämään myös mainosten kohdentamiseen sovelluksen käyttäjille Facebookissa.

4.7 Muut laajennukset

Edellä käsiteltyjen toiminnallisten laajennusten lisäksi Cordovalle on lukuisia yleis-hyödyllisiä laajennuksia. Network-information-laajennus seuraa verkon tilaa ja ilmoit-taa tilan muutoksista. Sovelluksessa sitä käytetään rajapintapyyntöjen yhteydessä tarkkailemaan, onko käyttäjä online-tilassa. Mikäli verkkoyhteyttä ei ole tai se kat-keaa pyyntöjen välissä, sovellus turvautuu offline-välimuistiin. Device-laajennus tun-nistaa laitteen käyttöjärjestelmän ja version. Optimitilanteessa sovelluslogiikka ei ole

riippuvainen järjestelmästä, mutta laajennuksesta on silti hyötyä. Sovelluksessa käyttöjärjestelmätietoa tarvitaan push-viestien rekisteröintiin ja lisäksi se vietiin HTML-dokumenttiin luokkana. HTML-luokan avulla tyylejä pystyttiin helposti kohdistamaan järjestelmän mukaan, esimerkiksi jakopainikkeissa järjestelmien omia jakosymboleita.

iOS-alustalle asennettiin wkwebview-engine-laajennus, joka ottaa käyttöön 8-version mukana julkaistun WKWebView-selainkomponentin. Cordova käyttää yhä oletuksena vanhaa UIWebView-komponenttia, joka ei ole enää täysin ajantasainen. Esimerkiksi näkymiä vierittäessä vanha komponentti pysäyttää käyttöliittymän ja vieritykseen pystyttiin reagoimaan vasta sen loputtua. Uuden komponentin saa käyttöön laajennuksella, mutta tällä hetkellä Cordovan virallisessa laajennuksessa on useita haasteita. Sovelluksessa käytetään Ionicin jatkokehittämää versiota, jossa useimmat haasteet on ratkaistu. (Cordova WKWebView Engine 2016.)

Näiden lisäksi käytössä olivat splashscreen- ja statusbar-laajennukset. Kun käyttöjärjestelmän kontrolloima latauskuva katoaa heti sovelluksen käynnistyttyä, splashscreen-laajennus jatkaa sen näyttämistä käyttöliittymäelementtinä ja kehittäjä voi itse piilottaa sen alustettuaan käyttöliittymän ja toteuttaa sulavamman siirtymän sovellukseen. Statusbar-laajennuksella pystytään kontrolloimaan käyttöjärjestelmän tilarivin näkyvyyttä sovelluksen käytön aikana.

5 Tulokset

Projektissa asiakkaalle toteutettiin mobiilisovellus ja sivustouudistus. Lopputulos vastasi asiakkaan määrittelyä ja kaikki suunnitellut toiminnot saatiin toteutettua. Projektin työmäärä pysyi pitkälle arvioidussa, mutta ylittyi hieman kommentointikierrosten yhteydessä. Projektin tekniseen toteutukseen arvioitiin 25 työpäivää ja lopulta siihen käytettiin noin 27,5 työpäivää, mikä oli 10 % ylitys arvioidusta. Aikataulullisesti projekti ei ollut kiireellinen, kirjoitusvaiheessa toteutus on luovutettu asiakkaalle sisällönsyöttöä varten ja odottaa julkaisua.

Projektissa toteutettiin myös useita ratkaisuja, joita toimeksiantaja pystyy hyödyntämään vastaavissa projekteissa. Mobiilisovellukseen toteutettiin yleispätevä prosessi

REST-rajapinnan hyödyntämiseen. Tekemällä rajapintapyynnöt keskitetysti ja tallentamalla ne LocalStorage-pohjaiseen välimuistiin, sovellus pystyi käyttämään online-sisältöä tehokkaasti riippumatta verkkoyhteyden nopeudesta tai tilasta. Samoin push-viestien käsittelystä toteutettiin esimerkkiratkaisu. Sovelluksessa otettiin push-viestit käyttöön käyttäjän luvalla, rekisteröitiin käyttäjän laite viestipalveluun ja käsiteltiin push-viestin yhteydessä saatua dataa. Lisäksi sovellukseen toteutettiin pohjaratkaisut natiivien jakotoimintojen käyttämiseen sekä sovelluksen käytön seurantaan Google Analyticsilla.

Myös WordPressin sisällönhallintaan tehtiin hyödynnettäviä ratkaisuja, kuten REST-rajapinnan käyttöönotto ja push-viestilaajennuksen valinta. Rajapinnan käyttöönotossa ratkaistiin myös omien sisältötyyppien ja ACF-laajennuksella tehtyjen tietokenttien lisääminen rajapintaan. Valitun push-viestilaajennuksen avulla sisällönhallinta hoitaa sekä laitetunnisteiden rekisteröinnin, että push-viestien lähettämisen saumattomasti sisältöpäivityksien yhteydessä.

Kokonaisuudesta saatiin myös toimeksiantajalle referenssi myynnin käyttöön. Keskitetysti ylläpidettävää sivusto- ja mobiilisovelluskokonaisuutta on helppo tarjota asiakkaille. Tässä vaiheessa referenssin perusteella on saatu yksi tarjouspyyntö vastaavasta toteutuksesta.

6 Pohdinta

Opinnäytetyön kirjallisessa osuudessa tarkasteltiin mobiilisovelluksen ja sivuston rajapinnan toteutusta. Työssä käsiteltiin ensin yleisesti Apache Cordovaa ja WordPressin REST-rajapintaa ja sen jälkeen mobiilisovelluksen keskeisiä ominaisuuksia.

Cordovan ohessa tutustuttiin myös muutamaan siihen pohjautuvaan kehitysratkaisuun ja niiden hyötyihin. Ionic on erinomainen ratkaisu natiivin käyttöliittymän kokoamiseen, mutta ei tarjonnut työkaluja sovelluksen testaamisen laitteilla. Monaca ja Adobe PhoneGap taas eivät ota kantaa käyttöliittymän toteutukseen, mutta tarjosivat testaustyökaluja sekä maksullisia lisäpalveluita. Monacalla oli kattavammat palvelut, joille ei kuitenkaan tässä projektissa ollut tarvetta. Sen palveluiden avulla olisi kuitenkin mahdollista toteuttaa vastaava sovellus ilman omaa sivustoa taustalla. Pro-

jektin kehitykseen valittiin PhoneGap, koska sen testaustyökalut olivat erittäin helppokäyttöiset ja natiiveja toimintoja, kuten push-viestejä pystyttiin testaamaan lennosta.

WordPressin sisäänrakennettu REST-rajapinta oli varsin helppo ottaa käyttöön. Asentamalla virallinen laajennus saatiin käyttöön valmiit päätepisteet, jotka kattavat lähes kaiken julkisen sisällön, myös omat sisältötyypit. Niiden lisäksi yhteisö on kehittänyt runsaasti laajennuksia, jotka laajentavat valmiiden päätepisteitä mahdollisuuksia. Valmiin ratkaisun kattavuus oli myös sen mahdollinen heikkous. Mikäli sisältöä on paljon, voi olla tarpeen toteuttaa omat suppeammat päätepisteet tiedonsiirron ja suorituskyvyn optimoimiseksi.

Sovelluksen arkkitehtuurista tuli hyvin suoraviivainen pääasiassa valittujen teknologioiden yhteensopivuuden ansiosta. Sivuston REST-rajapinnalta oli helppo hakea tarvittavat sisällöt ja rajapinnan JSON-muotoinen data pystyttiin tallentamaan keskitetysti offline-käyttöä varten sovelluksen välimuistiin. Data saatiin vietyä saumattomasti käyttöliittymälle käyttämällä Ractive.js:n HTML-pohjia ja automaattista datan peilausta. Varsinaisessa sovelluslogiikassa pääpaino pysyi datan ja tapahtumien käsittelyssä sekä varsinaisten määriteltyjen toimintojen toteutuksessa.

Vaikkei sovellusta vielä julkaistu, on julkaisuprosessista aikaisempaa kokemusta ja sovelluskauppoja käytettiin sovelluksen testijakeluun. Sovellus on tarkoitus julkaista Applen App Storeen ja Google Play -kauppaan. Aiemmissa julkaisuissa haastavinta on ollut aikataulutus, koska palveluntarjoajien hyväksyntäprosessit kestävät vaihtelevasti, eivätkä välttämättä mene läpi ensimmäisellä yrityksellä. Sovellus on kuitenkin jo päässyt läpi Applen esihyväksynnästä testijakelun yhteydessä. Applen hyväksyntäajat ovat myös laskeneet tänä vuonna keskimäärin alle kolmeen vuorokauteen (iOS App Store - Rolling Annual Trend Graph 2016). Googlen automatisoitu hyväksyntäprosessi vie kokemuksen perusteella alle vuorokauden.

Uskoisin, että sovelluksen julkaisu tulee onnistumaan ilman suurempia ongelmia. Joka tapauksessa projekti on omalta osaltani auttanut suoraviivaistamaan hybridisovellusten kehitystä ja tuonut tietämystä vaihtoehtoisista ratkaisuista ja työkaluista myös tuleviin projekteihin.

Lähteet

Adding Capabilities. 2016. Artikkeli Apple Developer -sivustolla. Viitattu 8.12.2016. https://developer.apple.com/library/content/documentation/IDEs/Conceptual/AppDistributionGuide/AddingCapabilities/AddingCapabilities.html

App Store Review Guidelines. N.d. Artikkeli Apple Developer -sivustolla. Viitattu 12.11.2016. https://developer.apple.com/app-store/review/guidelines/#minimum-functionality

Common Problems. 2016. Artikkeli WP REST API -sivustolla. Viitattu 17.8.2016. http://v2.wp-api.org/guide/problems/

Condon, E. 2016. Advanced Custom Fields. Viitattu 17.8.2016. https://wordpress.org/plugins/advanced-custom-fields/

Cordova Overview. 2016. Artikkeli Apache Cordova -sivustolla. Viitattu 8.12.2016. https://cordova.apache.org/docs/en/latest/guide/overview/index.html

Cordova WKWebView Engine. 2016. Artikkeli Github-sivustolla. Viitattu 6.12.2016: https://github.com/driftyco/cordova-plugin-wkwebview-engine

Cruz, R. S. 2015. Ractive-touch. Viitattu 8.12.2016. https://github.com/rstacruz/ractive-touch

Glossary. 2015. Artikkeli WP REST API -sivustolla. Viitattu 8.12.2016. http://v2.wp-api.org/glossary.html

Harris, R. 2016. Mustaches. Viitattu 8.12.2016. http://docs.ractivejs.org/latest/mustaches

Harris, R. 2016. Ractive.js - Next-generation DOM manipulation. Viitattu 2.12.2016. https://github.com/ractivejs/ractive

Ionic Component Documentation. N.d. Artikkeli Ionic Framework -sivustolla. Viitattu 1.12.2016. http://ionicframework.com/docs/v2/components/

iOS App Store - Rolling Annual Trend Graph. 2016. Tilasto Average App Store Review Times -sivustolta. Viitattu 8.12.2016. http://appreviewtimes.com/ios/annual-trend-graph

Monaca Backend. N.d. Artikkeli Monaca-sivustolla. Viitattu 6.12.2016. https://docs.monaca.io/en/manual/backend/

Mullenweg, M. 2015. WordPress 4.4 “Clifford”. WordPress.org 8.12.2015. Viitattu 12.11.2016. https://wordpress.org/news/2015/12/clifford/

Platform Support. 2014. Artikkeli Apache Cordova -sivustolla. Viitattu 17.8.2016. http://cordova.apache.org/docs/en/latest/guide/support/index.html

Products. N.d. Artikkeli Adobe PhoneGap -sivustolla. Viitattu 17.8.2016. http://phonegap.com/products/

Push Notifications. 2016. Artikkeli Adobe PhoneGap -sivustolla. Viitattu 6.12.2016. http://docs.phonegap.com/tutorials/develop/push-notifications/

Set up a GCM Client App on Android. 2016. Artikkeli Google Developers -sivustolla. Viitattu 8.12.2016. https://developers.google.com/cloud-messaging/android/client

Storage. 2016. Artikkeli Apache Cordova -sivustolla. Viitattu 20.11.2016. https://cordova.apache.org/docs/en/latest/cordova/storage/storage.html#localstorage

Usage of content management systems for websites. 2016. Tilasto W3Techs Web Technology Surveys -sivustolla. Viitattu 20.11.2016. https://w3techs.com/technologies/overview/content_management/all

Verbruggen, E. 2016. PhoneGap Social Sharing plugin for Android, iOS and Windows Phone. Viitattu 3.12.2016. https://github.com/EddyVerbruggen/SocialSharing-PhoneGap-Plugin/

Wheeler, K. 2016. Slick. Viitattu 8.12.2016. https://github.com/kenwheeler/slick